УДК 004.031.42

БОЛДАК А.О.,
ПУСТОВИТ М.А.

АРХИТЕКТУРА ИНФОРМАЦИОННОЙ СИСТЕМЫ ИНТЕЛЛЕКТУАЛЬНОЙ ОБРАБОТКИ ДАННЫХ

В работе сформулированы требования, которым должна удовлетворять информационная система коллективного пользования для интеллектуальной обработки данных. На основе этих требований определены архитектурные принципы и программные механизмы, составляющие основу программного каркаса для построения таких систем. Использование предложенной архитектуры обеспечивает повышение эффективности разработки и технического сопровождения информационных систем коллективного пользования, предназначенных для обработки больших информационных массивов.

In the article formulated the requirements which the information system of collective use for data mining has to satisfy. On the basis of these requirements are defined architectural principles and software mechanisms that form the basis of software framework for building such systems. Using the proposed architecture enhances the efficiency of development and technical support information systems for collective use, designed for processing large data arrays.

Введение

В последние десятилетия в научной среде все большее значение приобретает компьютерное моделирования и эксперименты, поставленные с его помощью. Область вычислительной науки (computational science)[1], занимающаяся проблемами создания инструментов для проведения подобных экспериментов так же бурно развивается. В некоторых случаях компьютерное моделирование выигрывает у натурного эксперимента по стоимости и по времени проведения эксперимента, дает более точные результаты.

Для проведения компьютерного моделирования сложных систем и анализа полученных результатов требуются значительные вычислительные мощности и объемы памяти для хранения данных. Актуальным становится решение задачи создания общедоступных информационных систем, которые могли бы использоваться большим количеством пользователей для проведения интеллектуальной обработки [2] больших информационных массивов.

Существующие сегодня решения подобного назначения, такие как Google Public Data Explorer [3] и Gapminder [4] не решают поставленную задачу в полной мере. Они являются закрытыми коммерческими системами, которые специализируются на визуализации данных. Расширение функциональности этих систем, использование непредусмотренных разработчиками источников данных и добавление новой функтиональности не представляется возможным.

Таким образом, цель настоящей работы состоит в повышении эффективности разкаботки и технического сопровождения информационных систем коллективного пользования для интеллектуальной обработки данных за счет определения архитектурных принципов, программных механизмов и информационных технологий, определяющих программные каркасы для построения таких систем.

1. Постановка задачи

Система статистической обработки данных должна удовлетворять требованиям, которые условно можно разделить на две категории. Одна из них связана с обеспечением требований, предъявляемых пользователем. Вторая же определят возможность быстрой разработки и компоновки как системы в целом, так и её компонентов.

С точки зрения пользователя такая система должна удовлетворять следующим требованиям:

- быть способной обрабатывать большие объемы информации и обрабатывать данные в «сыром» (raw data) необработанном формате;
- система должна быть доступна широкому кругу пользователей;
- система должна быть простой в использовании даже для пользователей, не знакомых с программированием;

- система должна позволять пользователю обрабатывать данные такого объема, который превышает объем хранилищ данных его оборудования.

Системные требования состоят в следующем:

- система должна быть масштабируемой (обеспечивать увеличение вычислительных ресурсов и объёмов хранения данных в соответствии с нагрузкой);
- внутренние механизмы обработки должны быть скрыты от пользователя. Они могут содержать алгоритмы являющиеся ноу-хау автора или использовать информацию, которая не находится в свободном доступе;
- архитектура и программные механизмы системы должны предусматривать возможность добавления новой функциональности и (или) реконфигурации системы.

2. Анализ современных программных технологий и средств разработки систем обработки даннях

Возможное превышение ресурсов, необходимое для обработки данных, над возможностями пользовательского оборудования заставляет использовать в качестве одного из архитектурных принципов организации системы её разделение на клиентские и серверные компоненты. Такая организация называется *клиент-серверной* архитектурой, при применении которой обработка данных производится на некоторых, возможно, удаленных, серверах [5]. В этом случае клиент только совершает запросы и получает результаты вычислений, т.е. оборудование пользователя играет роль тонкого клиента.

При этом могут использоваться сетевые протоколы уровня приложений и сеть Internet как коммуникационная среда. Это обеспечивает возможность использования разрабатываемой системы широким кругом пользователей.

Облачные вычисления. Использование клиент-серверной архитектуры с комуникационной средой на основе сети Internet позволяет определить разрабатываемую систему в терминах облачных вычислений, то есть парадигмы, в рамках которой информация постоянно хранится и обрабатывается на удаленных серверах в интернете и только временно кэшируется на клиентской стороне [6].

Существуют множество взглядов пользователя на системы облачных вычислений. Ему может предоставляться аренда аппаратных средств, виртуальная вычислительная среда или доступ к функционально законченному программному продукту.

Система статистической обработки данных с точки зрения пользователя должна выглядеть как *Software as a Service* (SaaS), которая предоставляет неподготовленному пользователю систему с простым и понятным интерфейсом [7]. При этом пользователь не должен заботиться, о поддержке работоспособности службы и развертывании ее на собственном оборудовании.

Выбор концепции облачных вычислений скрывает механизмы работы инструментов обработки данных, что является одним из требований к разрабатываемой системе.

Сервис-ориентированная архитектура. Принципы облачных вычислений могут быть реализованы в системах с различными архитектурами [8]. Одной из таких архитектур является *Сервис-ориентированная архитектура* (англ. SOA, service-oriented architecture), суть которой состоит в модульном подходе к разработке программного обеспечения, основанный на использовании сервисов (служб) со стандартизированными интерфейсами. SOA подразумевает множество слабосвязанных (либо вообще не связанных) служб, которые объединены общим коммуникационным механизмом.

В такой системе передача запросов, данных и служебной информации, происходит посредством коммуникатора. Каждая из служб занимается решением только одной задачи. Это позволяет многократно использовать службы для решения различных прикладных задач.

Сервисная шина предприятия. Одним из путей реализации SOA является использование механизма, называемого Enterprise Service Bus (ESB). Под данным понятием подразумевают, в некоторых случаях архитектурное решение, в других – набор программного обеспечения (ПО), решающего задачи взаимодействия компонентов системы. С точки зрения разработчика системы удобнее рассматривать ESB как набор ПО [9].

Для пользователя системная шина реализует концепцию единой точки доступа к приложению. Это подразумевает, что создается диспетчер, который несет ответственность за проведение запросов и выдачи результатов [11].

Таким образом, в архитектуру системы вводится промежуточное звено (шина), которое организует взаимодействие между подключаемыми к нему службами. При этом способы подключения служб к шине стандартизированы. Такая реализация хорошо масштабируется и расширяется.

Сегодня существует достаточно большое количество программных комплексов реализующих ESB (к примеру, IBM WebSphere ESB [12], JBoss Enterprise SOA Platform [13] и др.), что свидетельствует о целесообразности и эффективности описанного подхода к организации взаимодействия в слабосвязанных программных системах.

Одним из способов реализации подсистем для ESB являются веб-службы (англ. web service), которые представляют собой программные системы, идентифицируемые строкой URI, чьи общедоступные интерфейсы определены на языке XML [10]. Описание такой веб-службы может быть найдено другими подсистемами, которые могут взаимодействовать с ней, используя механизм передачи XML-сообщений. Эти сообщения передаются при помощи интернет-протоколов уровня приложений. Подобные протоколы, которые можно разделить на две категории: протоколы, основанные на использовании доступа к информационным ресурсам и протоколы, основанные на удаленном вызове процедур (RMI – remote method invocation).

Первую категорию называют REST (Representational State Transfer) протоколами. В случае использования этой архитектуры агенты пользователей взаимодействуют с ресурсами, которыми может быть всё, что можно поименовать и представить. Взаимодействие осуществляется с помощью единого интерфейса стандартных команд HTTP (GET, POST, PUT и DELETE). Для взаимодействия важно также объявить тип мультимедийного ресурса, который указывается с помощью заголовка типа содержимого HTTP. В этом случае вся информация, необходимая для обработки запроса ресурса, содержится в нем самом [14].

Протоколы второй категории используют механизм удаленного вызова процедур RMI (Remote Method Invocation), среди которых наиболее распространенным является протокол SOAP (Simple Object Access Protocol). Он предполагает передачу структурированных сообщений на основе XML. В отличие от REST, SOAP – это защищенный протокол с гарантированной передачей сообщений [15].

Каждый из описанных протоколов имеет свои достоинства и недостатки. Для обеспечения возможности интеграции системы статистической обработки данных с другими системами, необходимо реализовывать не один, а несколько протоколов, с использованием которых осуществляется передача сообщений.

3. Использование шаблонов проектирования для разработки программных механизмов системы статистической обработки данных

При реализации архитектурного решения ESB для системы статистической обработки данных ключевым аспектом является эффективная реализация программных механизмов, которые обеспечивают унификацию интерфейсов подсистем, методов доступа к данным, а так же их кеширование и визуализацию. При реализации этих программных механизмов могут быть использованы типовые решения уровня структурного проекттирования – шаблоны.

Унификация интерфейса подсистем. Общий интерфейс (API - application programming interface), который предоставляет система, не должен содержать всех функций, предоставляемых подсистемами, поскольку среди них могут быть утилитарные системные функции. Для скрытия внутренней структуры и создания общего интерфейса применяют шаблон Фасад [16, с.183], который можно использовать на двух уровнях: на уровне каждой подсистемы, в случае если она состоит из нескольких блоков и на уровне системы в целом. Во втором случае фасад представляет API системы.

Общий вид механизма интеграции подсистемы приведен на **Ошибка! Источник ссылки не найден.**.

ESB часто внедряется в системы, в которых уже созданы решения для многих задач. Но части системы, в которых реализованы решения, могут быть не выделены в отдельные структурные блоки. Даже если разделение на блоки уже произведено, их необходимо оформить в виде веб-служб со стандартизированными интерфейсами. В этом случае целесообразно использовать шаблон Адаптер [16, с. 141]. В этом случае в роли клиента выступает Диспетчер (ESB), в роли адаптируемого объекта – реализация службы.

Визуализация данных. Визуализация объектов это дополнительная функциональность,

которую удобно выносить в отдельные специализированные подсистемы. В таких случаях применяют шаблон Декоратор [16, с.173], цель применения которого состоит в добавлении функциональности объекту без порождения новых классов. Местоположение декоратора в общей структуре системы показано на **Ошибка! Источник ссылки не найден.**.

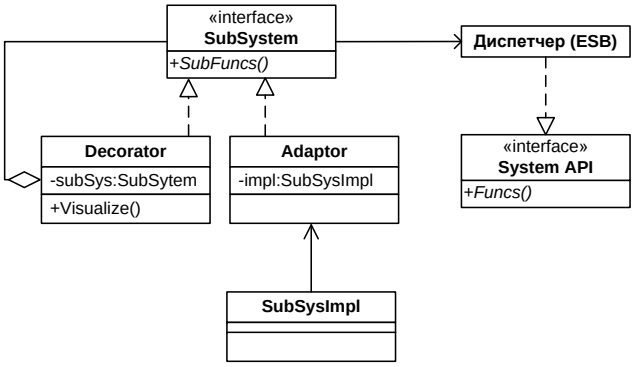

Рис. 1. Адаптер для интеграции подсистемы и декоратор для визуализации данных

Унификация методов доступа к данным. Источники данных, с которыми должна работать система, могут значительно различаться по устройству. Для обеспечения единообразного способа доступа к хранилищам данных используется шаблон Мост [16, с. 152] **Ошибка! Источник ссылки не найден.**, который

предусматривает разделение абстрактного интерфейса и реализации методов доступа. Следствием такого разделения является возможность создания нескольких реализаций, которые с точки зрения системы будут неотличимы в смысле протокола взаимодействия с источником данных.

Кеширование данных. Необходимость передачи и обработки больших массивов информации заставляет использовать специальные механизмы, которые обеспечивают минимизацию этих затрат. К таким механизмам относится кеширование данных и результатов запросов, реализация которых возможна с использование шаблона Заместитель (Proxy) [16, с. 203], (см. **Ошибка! Источник ссылки не найден.**). При использовании данного паттерна, тяжеловесный объект замещается объектом-заместителем. В данном случае в роли тяжеловесного объекта может выступать либо объект, хранящийся в удаленном хранилище данных, либо объект, выполнение функций которого может занимать значительное время. Заместитель ведет учет всех запрошенных данных и выполненных запросов на обработку, если происходит повторный запрос, то запрашивающей стороне отдается локальная копия. Это экономит вычислительные ресурсы и ресурсы сети.

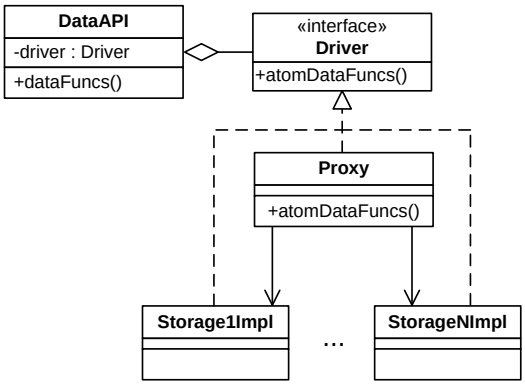

Рис. 2. Мост для создания API доступа к данным и заместитель для кеширования данных и запросов

4. Программный механизм обработки запросив

Координация работы подсистем реализуется диспетчером при условии, что эти подсистемы проектируются независимо друг от друга как самостоятельные функциональные блоки. При реализации такого диспетчера целесообразно использовать шаблон Посредник (Mediator) [16, с. 263].

Как видно из Рис. 3, на котором приведен пример последовательности действии при обработке запроса, вся ответственность за выполнение запроса возложена на Диспетчер

Концепция шаблона заключается во включении в систему объекта кординирующего работу ее частей. В этой системе обязанности посредника возлагаются на Диспетчер. При применении такого подхода, все подсистемы проектируются независимо друг от друга, как самостоятельные функциональные блоки.

Следует отметить, что в роли Клиента, может выступать не только пользователь, но и любая из подсистем.

Регистрация подсистем. В предложенной архитектуре диспетчер, играющий интеграционную роль, должен обладать функциональностью, обеспечивающей возможность расширения функциональности системы и (или) её реконфигурации. В этом случае элементами являются подсистемы в виде веб-сервисов.

Таким образом, диспетчер должен сохранять информацию о конфигурации системы. Реконфигурация системы или её функциональное расширение связанно с изменением этой информации.

Последовательность действий при регистрации подсистемы представлена на *Рис. 2.*

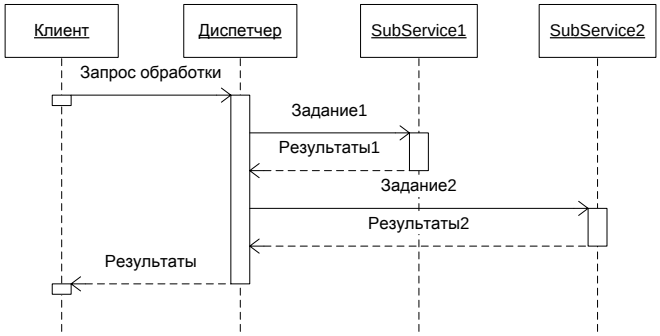

Рис. 3. Диаграмма последовательности выполнения запроса к системе

Для регистрации новой службы в системе пользователь должен направить соответструющий запрос диспетчеру, в котором указать URL (Uniform Resource Link) добавляемой системы.

Производится обращение к подсистеме с целью получения метаданных, описывающих подключаемую службу.

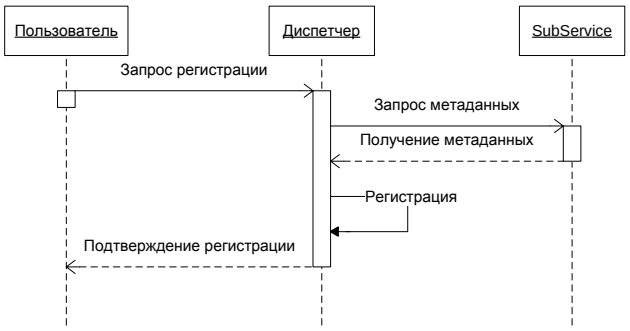

Рис. 2. Диаграмма последовательности регистрации новой службы в системе

После получения информации, диспетчер верифицирует её. Каждая служба может обрабатывать запросы нескольких типов, поэтому в метаданных системы должны содержаться сведения обо всех обрабатываемых типах запросов. Каждому типу должен быть присвоен уникальный URI (Universal Resource Identification) [17], состоящий из пути URL (совпадающий с URL добавляемой системы) и имени типа URN (Universal Resource Name). Диспетчер формирует таблицу маршрутизации запросов, которая используется им в процессе обработки запросов (см. Рис. 3).

Необходимым условием реализации описанной схемы является то, что API подсистемы обязательно должно содержать функцию, отдающую по запросу информацию, описывающую подсистему.

Заключение

В работе сформулированы требования, которые предъявляются различными категориями заинтересованных лиц, к информационной системе коллективного пользования для интеллектуальной обработки данных. Исходя из этих требований, определены принципы архитектурной организации и программные механизмы, составляющие программный каркас для построения такой системы. Эти принципы состоят в следующем.

Установлено, что выбор модели облачных вычислений удовлетворяет требованиям доступности системы и сокрытия механизмов обработки данных, что является необходимыми требованиями для обеспечения возможности использования системы широким кругом пользователей, не знакомых с программированием.

Использование принципов организации сервис-ориентированной архитектуры при реализации программного каркаса такой системы обеспечивает возможность ее масштабирования, а также расширения ее функциональности.

Основой программного механизма является диспетчер системы, реализующий функциональность системной шины предприятия (enterprise service bus), которая используется для организации взаимодействия всех компонентов системы. Это предполагает необходимость унификации интерфейсов подсистем, механизмов доступа к данным, кеширования и визуализации результатов их обработки.

Использование предложенной архитектуры информационной системы коллективного пользования для интеллектуальной обработки данных позволяет повысить эффективность разработки и технического сопровождения систем подобного назначения.

Список литературы

1. Syamlal, Madhava computational science: Enabling Technology Development [Электронный ресурс]// Syamlal, Madhava, Guenther, Chris, Cugini, Anthony, Ge, Wei, Wang, Wei, Yang, Ning, Li, Jinghai – Pe-

жим доступа: http://findarticles.com/p/articles/mi_qa5350/is_201101/ ai_n56829874/. Дата обращения: 28.04.2011
2. Christopher Clifton Data mining [Электронный ресурс] // C. Clifton – Режим доступа: http://www.britannica.com/EBchecked/topic/1056150/data-mining Дата обращения: 26.04.2011
3. Google Public Data Explorer [Электронный ресурс]// Режим доступа: http://www.google.com/publicdata/home. Дата обращения: 14.04.2011
4. Gapminder [Электронный ресурс]// Режим доступа: http://www.gapminder.org/. Дата обращения: 14.04.2011
5. Коржов В. Многоуровневые системы клиент-сервер [Электронный ресурс]// В. Коржов – URL: http://www.osp.ru/nets/1997/06/142618/#part_1. Дата обращения: 14.04.2011
6. Peter Mell, Tim Grance The NIST Definition of Cloud Computing // National Institute of Standards and Technology, Information Technology Laboratory - Version 15, 10-7-09
7. Колесов А. Модель SaaS – в мире и в России / А. Колесов // Журнал Byte Россия [Электронный ресурс]. - №10 (119), октябрь 2008 – Режим доступа: http://ej.kubagro.ru/plinks.asp
8. Сервис-ориентированная архитектура [Электронный ресурс] : Материал из Википедии – свободной энциклопедии : Версия 32077064, сохранённая в 14:44 UTC 19 февраля 2011 / Авторы Википедии // Википедия, свободная энциклопедия. – Электрон. дан. – Сан-Франциско: Фонд Викимедиа, 2011. – Режим доступа: http://ru.wikipedia.org/?oldid =32077064
9. Сервисная шина предприятия // Википедия. [2011–2011]. Дата обновления: 04.04.2011. Режим доступа: http://ru.wikipedia.org/?oldid=33340781 (дата обращения: 04.04.2011) http://www.w3.org/TR/ws-gloss/
10. Hugo Haas, Allen Brown Web Services Glossary [Электронный ресурс]/ Hugo Haas, Allen Brown. – Режим доступа: http://www.w3.org/TR/ws-gloss/ Дата обращения: 11.04.2011
11. Дональд Фергюсон, Марша Стоктон Модель программирования SOA для реализации Web-сервисов, Часть 1: Введение в модель программирования SOA [Электронный ресурс] // Фергюсон, Стоктон Режим доступа: http://www.ibm.com/developerworks/ru/ library/ws-soa-progmodel/
12. IBM WebSphere [Электронный ресурс]// Режим доступа: http://www-01.ibm.com/software/integration/wsesb/about/. Дата обращения: 14.04.2011
13. JBoss Enterprise SOA Platform [Электронный ресурс]// Режим доступа: www.jboss.com/pdf/SOA_infosheet.pdf. Дата обращения: 14.04.2011
14. Фландерс Д. Введение в службы RESTful с использованием WCF / Д. Фландерс // Журнал MSDN [Электронный ресурс]. – январь 2009 – Режим доступа: http://msdn.microsoft.com/ru-ru/magazine/dd315413.aspx
15. Маквитти Л. REST как альтернатива SOAP / Л. Маквитти // Сети и системы связи [Электронный ресурс]. Режим доступа: http://www.ccc.ru/magazine/depot/07_01/read.html?0502.htm
16. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. // СПб: Питер, 2001. – 368 с.
17. URI [Электронный ресурс] : Материал из Википедии – свободной энциклопедии : Версия 33228273, сохранённая в 21:27 UTC 30 марта 2011 / Авторы Википедии // Википедия, свободная энциклопедия. – Электрон. дан. – Сан-Франциско: Фонд Викимедиа, 2011. – Режим доступа: http://ru.wikipedia.org/?oldid=33228273